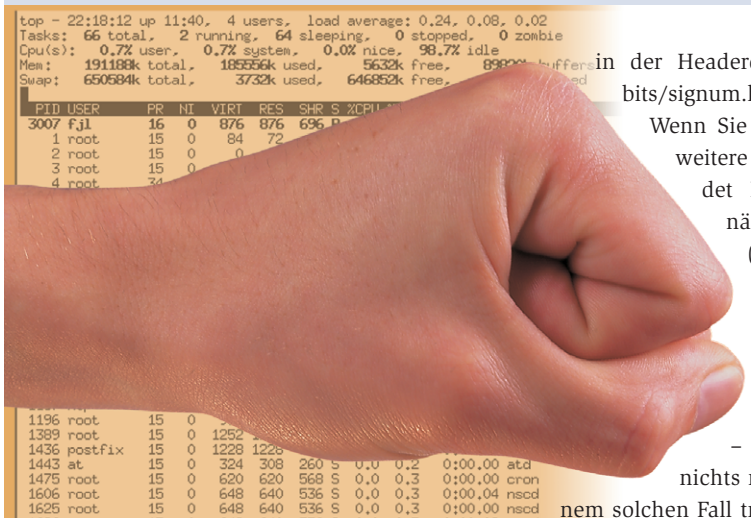
Aus dem Nähkästchen geplaudert: Interprozesskommunikation mit Signalen

Handzeichen

Auf einem typischen Unix-Computer verrichten mindestens 30 Prozesse gleichzeitig ihre Arbeit. Oft müssen sie miteinander kommunizieren. Die bekannteste Methode hierfür sind Signale, die auch Benutzer per »kill«-Kommando absetzen können. Marc André Selig



Ein Prozess startet, erledigt seine Arbeit, räumt auf und beendet sich. Bei vielen Aufgaben kann das eine ganze Weile dauern, oft über Wochen und Monate. Zum Beispiel ein Webserver: Er bleibt meist bis zum nächsten Update ununterbrochen im Speicher. Ähnliches gilt für die meisten Daemons. Die Umgebung solcher Software bleibt aber nicht immer gleich. Nach einer Korrektur in einer Konfigurationsdatei beispielsweise muss der Admin den zugehörigen Daemon auf die Änderung hinweisen. Oft ist auch eine Kommunikation zwischen Prozessen erforderlich.

Prozesse, hört die Signale

Zu den bekanntesten IPC-Techniken (Inter-Process Communication) zählen Signale. Jedes Mal, wenn Sie einen überflüssigen Prozess mit »kill *PID*« beenden, schicken Sie ihm in Wirklichkeit ein Signal (siehe Abbildung 1). In Abbildung 2 ist eine Liste der in »kill« definierten Signale zu sehen. Für Linux-Programme maßgeblich ist allerdings die Signalliste in der Headerdatei »/usr/include/bits/signum.h«.

Wenn Sie einen Prozess ohne weitere Angaben killen, sendet Ihr Kommando zunächst das Signal 15 (»TERM«, terminate). Freilich ignorieren manche Prozesse gerade dieses Signal oder sind bereits komplett abgestürzt – dann hilft »TERM« nichts mehr. Wer sich in einem solchen Fall trotzdem durchsetzen möchte, sagt »kill -9« und schickt damit das »KILL«-Signal: Kein Prozess kann es blockieren.

Wenn es sich also nicht gerade um einen Zombie (siehe Kasten „Zombies“) handelt oder um einen Prozess, der auf einen Ein- oder Ausgabevorgang wartet und deshalb im Kernelspace hängt, wird der Prozess sicher beendet.

Programme anhalten und fortsetzen

Von den übrigen Signalen sind einige besonders wichtig und kommen häufig zum Einsatz – das bleibt jedoch meist im Verborgenen. Bestimmt haben Sie schon mal einen Editor mit [Ctrl] + [Z] unterbrochen. Die Shell reagiert darauf mit:

```
[2]+  Stopped      vi meinedatei
```

Sie können den Editor später mit »fg« (foreground) wieder aktivieren.

Welche von der Shell kontrollierten Prozesse sich gerade in welchem Zustand befinden, erfahren Sie mit »jobs«:

```
[1]-  Running      emacs &
[2]+  Stopped      vi meinedatei
```

[Ctrl] + [Z] schickt das Signal 19, also »STOP«. Der Befehl »fg« setzt den unterbrochenen Prozess mit Signal 18 fort: »CONT« (continue). »bg« schickt ebenfalls das Signal 18, wobei dieses Kommando den Prozess zusätzlich in den Hintergrund verlegt, als hätten Sie ihn mit angehängtem »&« aufgerufen. Wenn Sie einen bereits im Hintergrund laufenden Prozess vorübergehend anhalten und später fortsetzen möchten, zum Beispiel weil er gerade zu viel Ressourcen verbraucht, können Sie auch direkt »kill -19« zum Anhalten und »kill -18« zum Fortsetzen verwenden.

Das obige Beispiel zeigt übrigens eine abgekürzte Notation der Prozessnummern, die die Bash als Erleichterung an-

Zombies

Startet ein Prozess einen anderen Prozess, spricht man von einer Mutter-Kind-Beziehung. Der aufrufende Prozess ist die Mutter, der aufgerufene das Kind. Stirbt das Kind vor seiner Mutter, erhält diese ein Signal namens »CLD« oder »CHLD« (child). Sie muss den Tod ihres Kindes sozusagen bestätigen und seinen Exit-Status (Rückgabewert) auslesen, üblicherweise durch eine »wait()«- oder »waitpid()«-Funktion.

Nur ein Eintrag in der Prozesstabelle

Tut sie das nicht, bleibt das Kind vorerst als Zombie [1] in der Prozesstabelle stehen. Der Kernel kann schließlich nicht wissen, ob beziehungsweise wann die Mutter den Exit-Status ihres Kindes doch noch abfragt, und muss ihn daher vorhalten.

Die meisten übrigen Ressourcen sind bereits freigegeben. Das ehemalige Kind existiert nicht mehr und lässt sich folglich nicht einmal von »kill -9« beeindrucken. Der Zombie verschwindet, sobald der Mutterprozess beendet ist oder den Exit-Status abfragt.

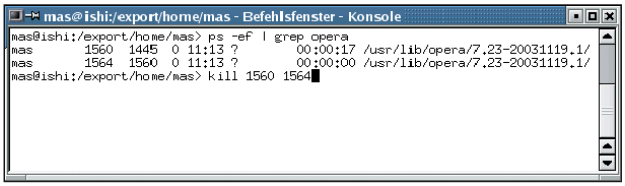


Abbildung 1: Der Benutzer »mas« sucht mit »ps« und »grep« nach Opera-Prozessen und fordert sie mit »kill *PID*« dazu auf, sich zu beenden.

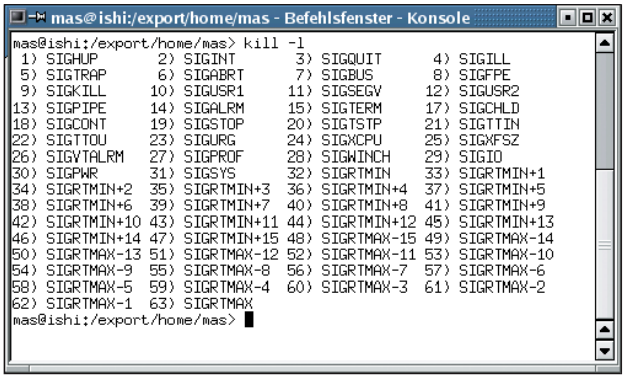


Abbildung 2: Linux kennt 64 verschiedene Signale, die neben ihren Nummern auch Namen tragen. Das verbreitete »kill -9« lässt sich daher auch als »kill -SIGKILL« oder »kill -KILL« schreiben.

bietet. Normalerweise erhält jeder Prozess unter Linux eine PID (Prozess-ID) im Bereich zwischen 1 und 32767. Die 1 ist für »init« reserviert, die Mutter aller Prozesse. Die ersten paar hundert PIDs verwendet der Kernel für eigene, die übrigen für Userspace-Prozesse.

PID-Auswahl bequemer

Es ist lästig, »kill -18 21967« wiederholt fehlerfrei eintippen zu müssen. Die Bash hilft: Sie kennt einen aktuellen und einen vorherigen Job, die sie in der Job-Übersicht mit einem Plus beziehungsweise einem Minus kennzeichnet. Wenn Sie ein Shell-Kommando wie »fg« benutzen, um einen Prozess zu manipulieren, dabei aber keine PID angeben, gilt das Kommando für den aktuellen Prozess. Zudem vergibt die Bash abgekürzte Prozessnamen, im obigen Beispiel »[1]« und »[2]«. Sie werden mit einem vorangestellten Prozentzeichen referenziert. Das Kommando »fg %1« hätte Emacs in den Vordergrund gerufen.

Zusätzlich zu [Ctrl] + [Z] (suspend) verstehen manche Shells auch den Delayed Suspend mit [Ctrl] + [Y]. Während Suspend sofort ein »STOP«-Signal schickt, wartet [Ctrl] + [Y] damit so lange, bis der Prozess Daten vom Terminal lesen möchte. So lässt sich ein Prozess punktgenau manipulieren, bevor Sie eine Eingabe vornehmen.

Ein anderes wichtiges Signal stammt aus der Zeit der seriellen Terminals: Signal

Nummer 1 heißt »HUP« oder Hangup und wird immer dann an einen Prozess geschickt, wenn am zugehörigen Terminal aufgelegt worden ist. Wer also über eine Modemleitung auf einem Unix-System arbeitet und die Telefonverbindung verliert, dessen Shell erhält ein »HUP«-Signal und kann aufräumen. Beispielsweise würde ein Editor noch schnell eine letzte Sicherungsdatei erstellen und sich dann erst beenden.

Auflegen auch ohne Modem

Modemleitungen sind heute eine Seltenheit, aber das »HUP«-Signal hat seine Bedeutung behalten. Es kommt beispielsweise zum Einsatz, wenn Sie eine SSH-Verbindung beenden. Eine problematische Situation ist in Abbildung 3a zu sehen: Benutzer »mas« wechselt zur Fernwartung auf eine fremde Maschine und startet einen aufwändigen Job. Auf das Ergebnis möchte er nicht warten, also schickt er den Job durch ein angehängtes »&« in den Hintergrund. Anschließend beendet er die Verbindung – was nicht auf Anhieb funktioniert, »ssh« scheint hängen geblieben zu sein. Also bricht er die Verbindung durch das SSH-Escape »~.« mit Gewalt ab.

In Wirklichkeit war die SSH nicht abgestürzt, sondern hatte den im Hintergrund wartenden Job bemerkt und hätte die Verbindung erst getrennt, wenn der Job abgeschlossen wäre. Durch den gewaltsamen Verbindungsabbruch erhielt der im Hintergrund laufende Prozess das »HUP«-Signal – und wurde damit ebenfalls beendet.

Besser wäre ein Vorgehen wie in Abbildung 3b. Der »nohup«-Befehl schirmt ein im Hintergrund ausgeführtes Programm vor dem Hangup-Signal ab. SSH bemerkt das zwar nicht und wartet wieder, doch dieses Mal bleibt das gewaltsame Trennen der Verbindung ohne negative Folgen: »dbdump« läuft friedlich weiter und kann beim nächsten Login seine Ergebnisse präsentieren.

Noch besser wäre es, den »dbdump«-Aufruf so abzusetzen:

```
nohup sudo -u mysql dbdump </dev/null &
```

Damit hat der Prozess keine Verbindung mit dem Terminal mehr und SSH beendet sich nach dem Logout, ohne dass Sie es mit »~.« erzwingen müssen.

Konfiguration geändert

In fast missbräuchlicher Weise interpretieren Daemons das Hangup-Signal. Hier ist die Auflegen-Analogie sinnlos – einem Daemon ist kein Terminal zugeordnet, folglich kann auch niemand die Verbindung trennen. Gemäß einer verbreiteten Konvention verwendet man »HUP« hier dazu, um den Daemon über Änderungen in seiner Konfigurationsdatei zu informieren.

Ein typisches Beispiel ist der Syslog-Daemon [2]. Im Regelfall soll er nur Warn- und Fehlermeldungen protokollieren, um den Speicherplatz auf der Festplatte und die Nerven des Administrators zu schonen. Nach der Installation eines neuen Softwarepakets ist aber mehr gefordert: Funktioniert nicht alles wie erhofft, wünscht man sich nähere Hinweis- und Debugging-Meldungen. Eine zusätzliche Zeile in »/etc/syslog.conf« lässt Syslog dies alles sammeln:

```
*.*    -/var/log/everything
```

Um »syslogd« auf die Änderung hinzuweisen, müssen Sie ihm ein »HUP«-Signal schicken. Mit »ps -ef | grep syslog« ist die PID des Syslog-Daemon schnell ermittelt, das Kommando »kill -1 *PID*« sorgt für den Rest. Wer sich die Zeit für das Heraussuchen der passenden PID sparen möchte, kann bei manchen Linux-Distributionen auch den Befehl »killall« benutzen, der ein Signal an alle Prozesse mit dem angegebenen Namen sendet:

```
killall -1 syslogd
```

Benutzerdefinierte Signale

Andere Signale, vor allem 10 (»USR1«) und 12 (»USR2«), stehen für benutzerdefinierte Aufgaben bereit. Wenn Sie die Konfigurationsdatei von Apache verändern, können Sie das erneute Einlesen durch ein »HUP«-Signal forcieren. Dabei nehmen Sie allerdings in Kauf, dass eventuell einige laufende Anfragen abgebrochen werden. Schicken Sie stattdessen ein »USR1«-Signal, so wird der Apache-Mutterprozess die einzelnen Kinder erst dann beenden und neu starten, wenn sie ihre gerade laufende Anfrage abgearbeitet haben.

Signale sind eine häufig eingesetzte und im täglichen Leben des Admins unentbehrliche Variante für die Kommunikation mit und zwischen den Prozessen eines Unix-Systems. Sie unterliegen jedoch gewissen Einschränkungen. Beispielsweise ist die Zahl der verfügbaren Signale beschränkt, sodass sie sich nur für die Übertragung weniger vordefinierter Hinweise eignen.

Aus Sicherheitsgründen darf zudem nur der Benutzer Root Signale an fremde Prozesse schicken. Ein Endanwender darf also nicht über Signale mit dem Mailserver kommunizieren. Aus diesem Grund gibt es zahlreiche alternative Mechanismen für die Interprozesskommunikation. Aber das ist bereits genug Stoff für eine künftige Folge der Admin-Workshop-Reihe. *(fjl)*

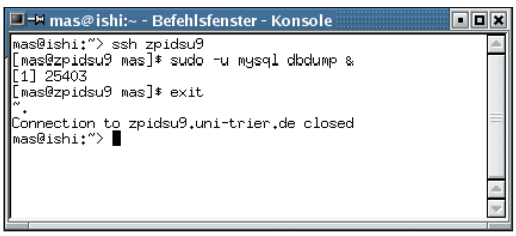


Abbildung 3a: SSH wartet auf das Ende aller Kindprozesse, bevor es die Verbindung tatsächlich beendet. Die Eingabe »~.« bricht die Verbindung vorzeitig ab.

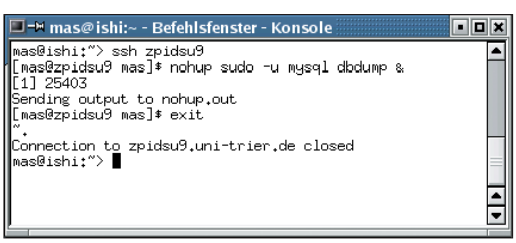


Abbildung 3b: Nur wenn lang laufende Hintergrundprozesse mit »nohup« gestartet wurden, überleben sie einen gewaltsamen Verbindungsabbruch mit »~.«.

Infos

[1] Wikipedia-Eintrag zu Zombie-Prozessen: [http://en.wikipedia.org/wiki/Zombie_process]

[2] Marc André Selig, „Computer-Logbuch: Syslog“: Linux-Magazin 02/04, S. 64.